ISSN : 2494-9150 Vol-02, Issue 01, APR 2016.

An Application Based on Hadoop Distributed File System

[1]Mirza Anjum, [2]Shaikh Asma, [3]Shaikh Sana, [4]Shaikh Ishrat, [5]Prof. Y. S. Pagar,
[1,2,3,4]BE-CSE Student, [5]Assistant Professor
[1,2,3,4,5]People's Education Society's College of Engineering, Aurangabad, Maharashtra, India.
[1]i.anjummirza@gmail.com, [2]asma.shaikh.171294@gmail.com, [3]sana.shaikh.25894@gmail.com,
[4]ishratshaikh8@gmail.com, [5]yspagar@yahoo.com

Abstract - In this paper we present the multi node installation of Hadoop cluster. We use this cluster to build an application which performs indexing and searching on large dataset. We use Apache Lucene framework alongside Hadoop to build a distributed search system. In today's world where information evolves rapidly, the huge growth of data across different geographical regions demands a system that assists fast parsing for the retrieval of meaningful results. A searchable index for distributed data would go a long way towards speeding the process. The searching on the data residing in Hadoop Distributed File System using indexing created by the Apache Lucene library. These indexes are then processed using Hadoop. To store very large datasets reliably Hadoop Distributed File System is designed. It is also able to stream those data sets at high bandwidth for the purpose of user applications. Distributed file system critically requires metadata management. All metadata is managed by a single server in the HDFS architecture, while file data is stored by a number of data servers.

Keywords* — *Apache Lucene, Dataset, Hadoop, HDFS, Master, Slave.

I. INTRODUCTION

Hadoop is an open source framework owned by Apache written in entirely Java. It uses simple programming models that allow distributed processing of large sets of data within clusters of computers. An application of Hadoop framework works in an environment which provides distributed storage and computation across clusters of computers. Hadoop is designed so as to scale up from single server to thousands of machines, which individually offers local storage and computation. In the past few years, data is increasing rapidly such as like Facebook, Twitter, Google, Yahoo!, web crawler, etc. Usage of data exceeds beyond Tera bytes so to maintain such type of data the traditional databases are not sufficient to store data.

The Hadoop Distributed File System is a distributed (HDFS) is a distributed file system designed to run on commodity hardware. It has to runs on commodity hardware. The Hadoop Distributed File System is fundamentally similar to the existing distributed file systems. HDFS differs from other distributed file systems in that it is highly fault tolerant and it can be deployed on low cost hardware. High throughput access is provided by HDFS to data applications which are also suitable for large dataset applications. HDFS architecture consists of NameNode, DataNode and HDFS Client. The figure below shows the HDFS architecture. [1][5]

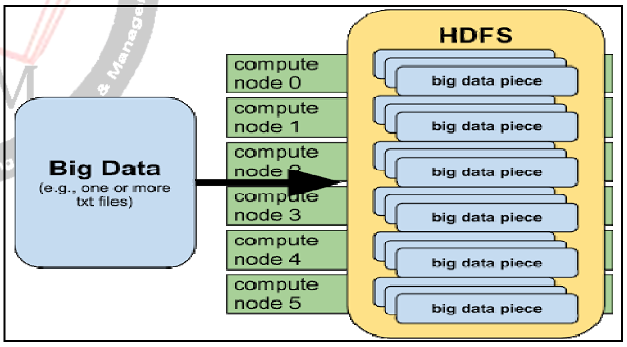


Figure 1: HDFS architecture

Lucene is a search engine library which belongs to Apache organization. It is free and open source software which is used to retrieve information. It is purely developed in Java. Search functionality is being added fast and furiously and since an open source nature is a big part of Hadoop ethos, the technology of choice in this case is often open source Lucene search engine. Hence indexing and searching on HDFS is done using Apache Lucene. It takes all the documents, splits them into words and then builds an index for each work. Since the index is an exact string-match, unordered it can be extremely fast. [2]

II. EXISTING SYSTEM

Microsoft SQL Server supports full text search since 1998, when version 7.0 was released. It is inbuilt feature of SQL Server, advantages of integration are provided by the system with RDBMS .

The main difference is within primary index structure. Unvarying database index is built on the whole field value, full text search of the procedure uses inverted index in its place. FTS goes through indexing the individual words within a text field in order to make searching through many records rapid Which is called tokenization .String search is still required within the field .So some of the organizations which are specific make available such benefits as "precision vs recall tradeoff", high search performance, stemming, ranking and many others usual FTS features. Index is created before starting searching, and after that indexes are updated automatically which are handled by SQL Server.

The users and applications run Full text queries alongside character based data which are present in SQL servers tables .Before running any full text queries on a table full text index must be created on the table by the database administrator .One or more character –based columns in the table are comprised in the full text index. These columns can have any of the following data types: FILESTREAM and **char**, **varchar**, **nchar**, **nvarchar**,**text**, , **image**, **xml**, or **varbinary(max)** Each full-text index indexes one or more columns from the table, Specific language can be used by individual column.

Linguistic searches are performed by full-text queries against text data in full-text indexes. The words and phrases are operated based on rules of a particular language such as English or Japanese. Full-text queries contain simple phrases or multiple forms of phrases. Documents are returned by full-text query that contain at least one match (also known as a *hit*). A match is displayed when a document which is targeted that contains all the terms that is specified in the full-text query, and meets any other search conditions, such as the distance between the matching terms.

1) *Full-Text Search Queries*

After columns have been added to a full-text index, users and applications can run full-text queries on the text in the columns. These queries can search for any of the following:

- One or more specific words or phrases (*simple term*)
- A word or a phrase where the words begin with specified text (*prefix term*)
- Inflectional forms of a specific word (*generation term*)
- A word or phrase close to another word or phrase (*proximity term*)
- Synonymous forms of a specific word (*thesaurus*)
- Words or phrases using weighted values (*weighted term*)

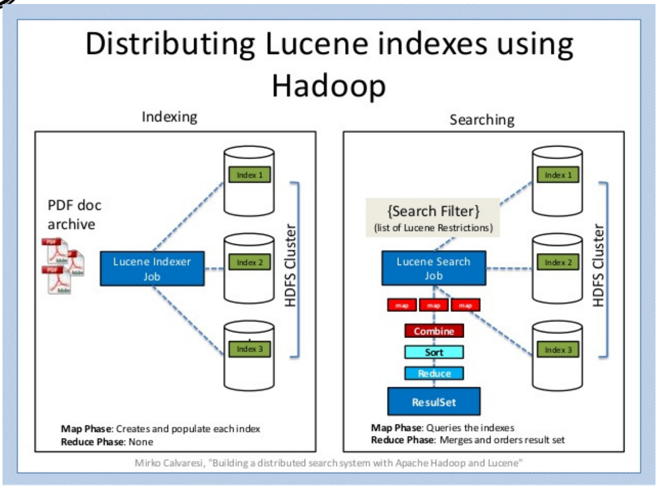
Full-text queries are not case-sensitive. For example, searching for "Aluminum" or "aluminum" returns the same results.
Full-text queries use a small set of Transact-SQL predicates (CONTAINS and FREETEXT) and functions (CONTAINSTABLE and FREETEXTTABLE). However, the search goals of a given business scenario influence the structure of the full-text queries. For example:

```
e-business—searching for a product on a website:
SELECT product_id
FROM products
WHERE CONTAINS(product_description, ”Snap Happy
100EZ” OR FORMSOF(THESAURUS,’Snap Happy’)
OR ‘100EZ’)
AND product_cost < 200 ;
Recruitment scenario—searching for job candidates that
have experience working with SQL Server:
SELECT candidate_name,SSN
FROM candidates
WHERE CONTAINS(candidate_resume,”SQL Server”)
AND candidate_division =DBA;
```

III. PROPOSED SYSTEM

Distributed search system using Lucene library

Indexing and searching on a HDFS is done using Apache Lucene. Lucene is a library which uses inverted full-text index technique. This implies that it takes as input all the documents, and splits them into words. It then builds an index for each word. Since the index is an exact string-match, unordered, it can extremely perform fast searching. A big index is created by Lucene. There are two entities present inside index which is work id and number of documents where the word is present. It also contains the position of words in those documents. So when a query is fired to search a single word the indexes are searched. The time complexity for this is O (1). Different algorithms are used to rank the results. In order to process the multi word query, it just takes the intersection of the set of files where the words have their location. Thus Lucene is very fast. We built a distributed search system using Hadoop and Lucene library. A searchable index for distributed data would go a long way towards speeding the process. The searching on the data residing in HDFS is performed using indexing created by Lucene library. These indexes are then processed in Hadoop. [3][4].


Distributing Lucene indexes using Hadoop
Indexing
Searching
PDF doc archive
Lucene Indexer Job
Index 1
Index 2
Index 3
HDFS Cluster
{Search Filter}
(list of Lucene Restrictions)
Lucene Search Job
map
map
map
Combine
Sort
Reduce
ResulSet
Map Phase: Creates and populate each index
Reduce Phase: None
Map Phase: Queries the indexes
Reduce Phase: Merges and orders result set
Mirko Calvaresi, "Building a distributed search system with Apache Hadoop and Lucene"

Our system can be used to perform searching on Big Data. The analyst can be benefited from our system as we provide a search interface to dig in a large dataset. The client, after performing search, is provided with all the matching records from the large dataset within a minute. Our system is suitable to work in organizations who deal with big data, where the analysts need to analyze the data like to find the number of records matching a particular keyword, to list and process them, etc.

IV. CONCLUSION

Deploying large datasets on HDFS can be easily configured commodity hardware, hence cutting down the cost of buying expensive servers or contracting cloud service providers for a PaaS setup. Using HDFS as a base, we can eliminate issues arising from lack of server scalability and infrastructure cost. Down times can be cut down significantly by the high availability features present in the system. By using a web interface the need for platform dependent applications is eliminated, allowing users to access the system on any machine within its organization's campus. This also eliminates the users need to understand the functionality involved in the backend. In our system we have added Lucene's index search capability into Hadoop. Lucene enables us to use the RAM directory for indexing and searching. The proposed system creates indexes for the file residing in HDFS and performs search using those indexes. The client has provided with a search interface. All the matching records are displayed in the browser. Thus, our system is a fast and efficient search engine system that provides better searching functionality than the traditional RDBMS approach.

We can safely conclude that ***this model is a successful implementation of the HDFS.***

V. FUTURE DEVELOPMENT SCOPE

The proposed system is an example of how the search functionality can be achieved through the combination of Hadoop and Lucene. The indexing and searching power of Lucene is combined with Hadoop's processing power. The Hadoop cluster we implemented consists of two nodes. To achieve faster processing and scalability, more data nodes can be added into the cluster. Lucene works with all kind of textual files. In our system we use CSV format files. Our system can work with different textual format. The dataset used in the proposed system consists of 15 columns and upto 44, thousand records. The dataset can be extended without any limit for performing search on larger datasets. In order to fully evaluate the expected benefits of the proposed system there is a need to implement this system in real environment.

The proposed system can be improvised by using better system configurations having larger RAM, more processor speed, more disc space, etc. The system proposed has the potential to be widely applicable for future use.

ACKNOWLEDGMENT

We would like to thank all members of the HDFS team at Yahoo! present and past for their hard work building the file system. We would like to thank all Hadoop committers and collaborators for their valuable contributions.

REFERENCES

[1] "A study on Digital Library using Hadoop Distributed File System", IJIRAE 2015

[2] "The Alexandria Digital Library Architecture", ECDL paper

[3] "The Hadoop Distributed File System paper", IEEE

[4] "Hadoop in Action" Chuck Lam

[5] "Lucene in Action 2nd Edition", Michael McCandless.